Course reader: *Debugging*

- Errors are inevitable. Don't let them get you down!

- Difficulties in MATLAB can be grouped into three categories:
  1. *You know the function name but don't understand how it works*. This is the least annoying situation, because help files and the Internet have examples you can follow.
  2. *You know what you want Matlab to do, but you can't figure out the command or function for it*. One of the advantages of MATLAB is its huge world-wide user base. It's pretty likely that the same or similar problem has been solved. If not, post your question to the MATLAB help forum.
  3. *You know what you want Matlab to do and you know the command to do it, but there are errors when you run the code*. This produces the most frustration. When you get to this point, see the rest of this pdf...

- When you get an error, follow the following steps:
  1. **Find the code causing the error**. This is not always as simple as it sounds. The error might be in a subfunction or you might be running an entire cell and not know where the error is. Run the code line-by-line until you get an error, or use break-points or stop-on-error if it's a function.

     Even when you find the *line* that causes the error, you might still have to find the *piece of code* that causes the error. It could be a dense line of code that mostly is fine except for one variable. Consider, for example, the line of code below.
     `tf(chani,:,ei,3) = abs(mean(exp(1i*angledata(:,trials4analysis)),2))`
     It might take some additional work to figure out which part of that line is the cause of the error (more on this in the next bullet point).
  2. **Inspect the error message in the command window**. The messages are sometimes cryptic, but look for keywords:
     (a) "dimension mismatch": This is likely because you are trying to assign data into a variable of a different size, as in this example: `v(2:3) = randn(3,4);`
     (b) "index exceeds dimensions": This happens when trying to access elements of a matrix that don't exist: `v=[1 2]; v(3)`
     (c) "Subscript indices must be real...": Indices have to be non-zero counting numbers (1,2,3,...). Perhaps the code is trying to access the 1.5^{th} element or the -3^{rd} element. Keep in mind that MATLAB starts counting at 1, not at 0.
     (d) "function or variable does not exist": If you are trying to call a function, then perhaps it's misspelled or not in the MATLAB path. If you are trying to access a variable, then it hasn't been created (perhaps because an earlier `if` statement didn't run).
     (e) "Not enough input arguments" or "too many output arguments": Check the help file of the function and look for the description of mandatory inputs.

  3. In some cases, just finding the exact piece of code that caused the error will help you solve it. If you are still stuck, then your best friends are the `size` and `plot` functions. Check the sizes of all relevant variables. Plot (or image, or make bar plots, or whatever is the most useful visualization) all of the relevant variables. Very often, insights into problem-solving come from

looking at the data.

- When the error line has been found, try to break up a long line of code to short pieces. Let's go back to the code line earlier. If I would get an error on that line, below are the following codes I would run in the command window.

  ```
  tf(chani,:,ei,3) = abs(mean(exp(1i*angledata(:,trials4analysis)),2))
  size( tf(chani,:,ei,3) )
  size( abs(mean... ) % check that the sizes match
  trials4analysis
  angledata(:,trials4analysis)
  mean(exp(1i*angledata(:,trials4analysis)),2)
  ```

  And so on. Notice my strategy: I first check the sizes of the code on the left and right sides of the equals sign, and then proceed to run code starting from the innter-most part of the code going outwards. Typically, you would work one set of parentheses at a time.

- Be patient and methodical. Don't try to make one-off short-cuts. That might work on this occasion, but it's almost certain to come back and make your life harder in the future.

- Three tips to help prevent coding errors:
  1. Use meaningful variable names and script/function names.
  2. Write clean and efficient code.
  3. Initialize variables.

- You can use the function pair tic/toc to help you improve your code:

  ```
  tic
  % code version A
  t(1) = toc;

  tic
  % code version B
  t(2) = toc;
  ```

  If the outputs of code versions A and B are the same, then whichever is faster (the smallest element in `t`) is the preferred method.

- **Most important: Learn something!** Every mistake in an opportunity for improvement (in programming and in life more generally). Yes, it sucks to spend 4 hours debugging code only to find out that you simply misspelled a variable name if a conditional in an `if` statement. But if you learn from that mistake, next time you'll solve the problem in seconds instead of hours.

Exercises

1. Each of the following lines of code will produce an error. First, try to figure out what the error is. Next, run the code in MATLAB and inspect the error message. Finally, fix the error and think about how that error message might help you debug in the future.

```
asdf = linspace(-30:30);
cm=rand(1,10);  morty=cm(0);
tu(:,4) = nan(4,3);
[m1,m2] = mean([randn(1,10) zeros(5,1)']);
asdf=zeros(3,4);  asdF(3,2)
for i=1:3 tmp(i)=i end
bsxfun(@times,randn(5,2),linspace(1,5,4)')
cat(1,zeros(10,2),ones(10,3))
dsearchn(linspace(-10,10,100)',[3 4])
plot(1:10,(1:10).^2,log(1:10))
r=1:10; r(rand)
[randn(1,10) zeros(5,1)]
std(randn(1000,1),2)
imagesc(randn(5,3,2))
q(:,1) = bsxfun(@times,randn(5,2),linspace(1,5,2));
a=rand; clear; c=a;
e = reshape(randn(5,3,4),2,3);
bsxfun(@times,randn(5,2),linspace(1,5,5))
r=1:10; r(round(rand))
for i=-3:3, tmp(i)=i; end
plot(1:5,,5:9)
plot(1:5,5:1)
```

2. Initialize the matrix `sqmat` (1) before the for-loop and then (2) *inside* the loop.

```
nums2sq = -15:1.2:19;
for i=2:length(nums2sq)
   sqmat(i,1) = abs(nums2sq(i))^1.2 - mean(sqmat(i-1:i,1));
   sqmat(i,2) = nums2sq(i)^2 - sign(nums2sq(i))*log(abs(nums2sq(i)));
end
```

3. The variable `outdat` is initialized to be really big, but the code doesn't run through 100,000 iterations. Cut out unused rows after the `while` loop.

```
outdat = zeros(100000,3);
l = 50;
r = 1;
while l>1
   % computations here...
   outdat(r,:) = results;
   r = r+1;
```

```
    l = l*.973;
end
```

Answers

1. Each line is adjusted. Note that "failing to produce a MATLAB error" is *not* the same thing as "correctly doing what the programmer had in mind"!

```
asdf = linspace(-30,30,length(-30:30));
cm=rand(1,10);  morty=cm(1);
tu(:,4) = nan(4,1);
m1 = mean([randn(1,10) zeros(5,1)']);
asdf=zeros(3,4);  asdf(3,2)
for i=1:3, tmp(i)=i; end
bsxfun(@times,randn(5,2),linspace(1,5,5)')
cat(1,zeros(10,2),ones(10,2))
dsearchn(linspace(-10,10,100)',[3 4]')
plot(1:10,(1:10).^2,1:10,log(1:10))
r=1:10; r(ceil(rand))
[randn(1,10) zeros(5,1)']
std(randn(1000,1),0)
imagesc(randn(5,3,3))
q = bsxfun(@times,randn(5,2),linspace(1,5,2));
a=rand; c=a;
e = reshape(randn(5,3),5,3);
bsxfun(@times,randn(5,2),linspace(1,5,5)')
r=1:10; r(round(rand)+1)
for i=-3:3, tmp(i+4)=i; end
plot(1:5,5:9)
plot(1:5,5:-1:1)
```

2. Initialize before:

```
nums2sq = -15:1.2:19;
sqmat = zeros(length(nums2sq),2);
for i=2:length(nums2sq)
   ...
```

Initialize during (make sure to initialize only on the first iteration!

```
nums2sq = -15:1.2:19;
for i=2:length(nums2sq)
   if i==2 % first iteration only
      sqmat = zeros(length(nums2sq),2);
   end
   ...
```

3. One solution is to remove rows that contain all zeros (best not to test for zeros in only one row, in case there are zeros in valid data entries).

```
toremove = sum(outdat,2)==0; % boolean for rows that sum to zero
```

```
outdat(toremove,:) = [];
outdat(toremove) = [];% this line is WRONG! (make sure you know why)
```